Bazy Danych

Ćwiczenie 1 – podstawy

Podstawy języka zapytań SQL.

Ćwiczenie 1 – podstawy

Systemy Zarządzania Bazą Danych (SZBD) pozwalają na rozwiązanie wielu problemów związanych ze składowaniem, przeszukiwaniem i przekształcaniem danych w bazach danych. Poszczególne czynności, które mają zostać wykonane przez SZBD, programista opisuje w specjalnym języku. Celem tego ćwiczenia, jest zapoznanie państwa z podstawami języka SQL, który jest jednym z najbardziej popularnych języków baz danych, stosowanym w prawie każdym komercyjnym SZBD na rynku.

Wymagania:

Podstawowe wiadomości z algebry i języków programowania oraz znajomość podstaw modelu relacyjnego (znajomość i zrozumienie terminów relacja, atrybut, krotka i system zarządzania bazą danych).

Plan ćwiczenia

- Wprowadzenie do laboratorium.
- Podstawowe informacje o języku SQL.
- Najprostsze zapytania.
- Projekcja.
- Wyrażenia.
- Aliasy.

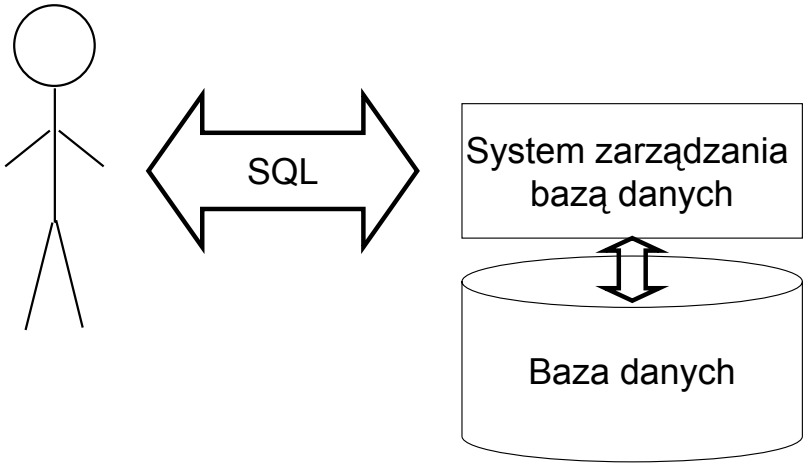
Ćwiczenie rozpoczniemy od wprowadzenia do laboratorium i przedstawienia podstawowych informacji o języku SQL. Po wstępie przejdziemy do omówienia składni polecenia SELECT pozwalającego na odszukiwanie danych w bazie danych. Zaczniemy od składni tego polecenia pozwalającej na wykonywanie najprostszych zapytań. Następnie powiemy co to jest projekcja i jak można ją wykonać za pomocą polecenia SELECT. Powiemy również jak można obliczać wyrażenia i jak nadawać wynikom wyrażeń nazwy (aliasy).

Plan ćwiczenia – cd.

- Wartości puste.
- Eliminacja duplikatów.
- Sortowanie wyników zapytania.
- Selekcja.
- Złożone warunki selekcji.
- Podsumowanie.

Następnie, omówimy problematykę związaną w wartościami pustymi, oraz omówimy składnię polecenia SELECT pozwalającą na eliminację duplikatów i sortowanie wyników działania polecenia. Na koniec ćwiczenia powiemy co to jest selekcja, jak można ją wykonać za pomocą polecenia SELECT, oraz omówimy sposoby konstrukcji złożonych warunków selekcji. Ćwiczenie zakończymy slajdem podsumowującym omawianą tematykę. Omawiane na zajęciach tematy będą ilustrowane zadaniami do samodzielnego wykonania.

Wprowadzenie do laboratorium



Ćwiczenia z przedmiotu „Bazy Danych” są poświęcone przede wszystkim językowi SQL (ang. *Structured Query Language*) i jego zastosowaniom. Język SQL jest strukturalnym językiem zapewniającym możliwość wydawania poleceń do systemu zarządzania bazą danych (SZBD). Dzięki temu językowi możliwe jest tworzenie poleceń, które nakazują SZBD odnaleźć potrzebne przez nas dane w bazie danych, jak również poleceń, dzięki którym można składowane dane zmodyfikować, wstawić nowe dane, bądź je usunąć. Dzięki temu językowi możliwe jest również definiowanie, modyfikacja i usuwanie struktur danych, w których dane są składowane. Język SQL pozwala również na zarządzanie transakcjami i mechanizmami autoryzacji dostępu do danych.

Na tym ćwiczeniu zapoznacie się państwo z podstawami pracy z językiem SQL, czyli z konstrukcją prostych zapytań do bazy danych. Wiedza ta będzie następnie rozbudowywana w kolejnych ćwiczeniach.

Język SQL

- SQL jest językiem deklaratywnym.
- Język SQL jest zorientowany na przetwarzanie zbiorów.
- Język SQL można podzielić na:
  - DML (ang. *Data Manipulation Language*)
  - DDL (ang. *Data Definition Language*)
  - DCL (ang. *Data Control Language*)
- W języku SQL nie rozróżnia się dużych i małych liter.
- W poleceniach SQL ignorowane są znaki końca linii.
- Każde polecenie SQL powinno być zakończone średnikiem.

Język SQL jest językiem deklaratywnym. Charakterystyczną cechą języków deklaratywnych jest to, że opisują „co” ma być zrobione, ale nie „jak”. W praktyce oznacza to, że użytkownik opisuje w języku SQL efekt jaki chce uzyskać (odczytanie telefonów wszystkich pracowników o nazwisku zaczynającym się na literę ‘A’, podniesienie pensji wszystkim profesorom itp.), ale nie sposób w jaki ma to być zrobione (sekwencja operacji dyskowych, które prowadzą do wykonania polecenia). Sposób wykonania polecenia jest automatycznie dobierany przez SZBD i zależy od fizycznego sposobu składowania danych.

Język SQL jest zorientowany na przetwarzanie zbiorów. Ponieważ dane są przechowywane w relacjach, które są zbiorami krotek, konstrukcje języka SQL dotyczą przetwarzania zbiorów i nie zawierają poleceń uwzględniających jakiś porządek na krotkach.

W języku SQL można wyróżnić trzy grupy poleceń:

DML – język manipulacji danymi pozwalający na odczytywanie danych z relacji (polecenie SELECT) oraz na wstawianie, modyfikację i usuwanie danych z relacji (polecenia: INSERT, UPDATE, MERGE i DELETE).

DDL – język definicji danych pozwalający na tworzenie, modyfikację i usuwanie relacji (polecenia CREATE, ALTER i DROP).

DCL – język kontroli danych pozwalający na zapewnienie autoryzacji dostępu do danych oraz zarządzanie transakcjami. Najważniejsze polecenia to GRANT i REVOKE (czasem zaliczane do DDLa) oraz COMMIT, ROLLBACK i SAVEPOINT.

W języku SQL nie rozróżnia się dużych i małych liter. Wszystkie słowa kluczowe i nazwy (np. relacji i atrybutów) w języku SQL można pisać zarówno dużymi jak i małymi literami. Przykładowo, wyrażenia: nazwisko, Nazwisko i NAZWISKO są identyczne.

W poleceniach SQL ignorowane są znaki końca linii. Wszystkie słowa kluczowe i wyrażenia w języku SQL można rozdzielać zarówno spacjami jak i znakami końca linii. W rezultacie dowolne polecenie SQL można sformatować w dowolny sposób (może ono zajmować jeden długi wiersz, lub kilka krótszych wierszy).

Każde polecenie SQL powinno być zakończone średnikiem. Próba wykonania polecenia nie zakończonego w ten sposób może zakończyć się błędem, chociaż istnieją sytuacje, gdy pominięcie średnika jest dozwolone.

Bazy danych

Schematy przykładowych relacji

ZESPOLY

ID_ZESP	NAZWA	ADRES

ETATY

NAZWA	PLACA_OD	PLACA_DO

PRACOWNICY

ID_PRAC	NAZWISKO	IMIE	ETAT	ID_SZEFA	.....

.....	ZATRUDNIONY	PLACA_POD	PLACA_DOD	ID_ZESP

Ćwiczenie 1 - podstawy (7)

Ćwiczenia z języka SQL zostaną przeprowadzone na bazie danych złożonej z trzech relacji przechowujących dane dotyczące pracowników uczelni wyższej. Relacje te to: ZESPOLY, ETATY i PRACOWNICY. Relacja ZESPOLY przechowuje dane dotyczące zespołów pracowników, którzy zajmują się różną tematyką naukową, bądź działalnością administracyjną. Każdy zespół posiada unikalny identyfikator (atrybut ID_ZESP), swoją nazwę (atrybut NAZWA) oraz adres (atrybut ADRES), pod którym znajdują się pomieszczenia zespołu. Relacja ETATY przechowuje dane dotyczące widełek płacowych pracowników na poszczególnych etatach. Atrybut NAZWA określa nazwę etatu, a atrybuty PLACA_OD i PLACA_DO określają jaka jest minimalna i maksymalna miesięczna płaca pracownika zatrudnionego na określonym etacie. Ostatnią relacją jest relacja PRACOWNICY, która przechowuje dane dotyczące pracowników uczelni. Kolejne atrybuty mają następujące znaczenie: ID_PRAC to unikalny identyfikator pracownika, NAZWISKO i IMIE, to odpowiednio nazwisko i imię pracownika, ETAT to nazwa etatu pracownika, ID_SZEFA to wartość identyfikatora pracownika, który jest bezpośrednim szefem pracownika opisywanego w danej krotce, ZATRUDNIONY jest datą zatrudnienia pracownika, PLACA_POD, to podstawa miesięcznej pensji pracownika, która bywa niekiedy rozszerzana o płacę dodatkową (atrybut PLACA_DOD). Ostatnim atrybutem jest ID_ZESP, który opisuje wartość unikalnego identyfikatora zespołu, do którego należy pracownik.

Relacje ZESPOLY, ETATY i PRACOWNICY można utworzyć za pomocą skryptu zawierającego polecenia SQL, o nazwie pracownicy.sql, który został załączony do kursu.

Proste zapytania

```
SELECT * FROM zespoly;
```

ID_ZESP	NAZWA	ADRES
10	ADMINISTRACJA	PIOTROWO 2
20	SYSTEMY ROZPROSZONE	PIOTROWO 3A
30	SYSTEMY EKSPERCKIE	STRZELECKA 14
40	ALGORYTMY	WIENIAWSKIEGO 16
50	BADANIA OPERACYJNE	MIELZYNSKIEGO 30

Naukę języka SQL zaczniemy od zapoznania państwa ze składnią polecenia SELECT. Jest to polecenie pozwalające na odczytywanie danych z bazy danych oraz wykonywanie na tych danych prostych obliczeń i przekształceń. Polecenie SELECT pobiera krotki z relacji w bazie danych, przetwarza je (opcjonalnie) i zwraca wynik w postaci zbioru odczytanych krotek. W wyniku wykonania polecenia SELECT otrzymujemy zatem relację (zbiór krotek), tzw. „relację wynikową”. Aplikacje klienckie, pozwalające na bezpośrednie wykonywanie poleceń SQL (np. psql w PostreSQL albo sqlplus w Oracle), przedstawiają relację wynikową w postaci tabelarycznej. Ponieważ polecenia SELECT służą do odczytywania danych w bazie danych, nazywa się te polecenia „zapytaniami”.

Najprostszą wersją polecenia SELECT, jest polecenie postaci: „SELECT * FROM {nazwa relacji};”. Polecenie odczytujące dane z relacji rozpoczyna się zawsze słowem kluczowym SELECT po którym podaje się listę atrybutów, które mają zostać odczytane. Sposób definiowania listy atrybutów zostanie opisany później. Wstawiona po słowie kluczowym SELECT gwiazdka oznacza „odczytaj wszystkie atrybuty”. Następnie, umieszcza się w poleceniu słowo kluczowe FROM, po którym podaje się nazwę relacji z której mają zostać odczytane krotki. Polecenie SELECT o postaci SELECT * FROM {nazwa relacji}; powoduje odczytanie wszystkich krotek i wszystkich atrybutów z relacji o podanej nazwie. Przykładowo, polecenie odczytujące całą zawartość relacji ZESPOLY wygląda następująco:

```
SELECT * FROM zespoly;
```

Powyższe polecenie można przetłumaczyć na język naturalny następująco: „Odczytaj wszystkie krotki z relacji o nazwie ZESPOLY, zachowując wszystkie atrybuty krotek (*)”. W wyniku takiego zapytania SZBD odczyta z bazy danych, z relacji ZESPOLY, wszystkie krotki i w postaci niezmienionej zwróci je aplikacji klienckiej, która w naszym przypadku wyświetli je na ekranie. Należy zwrócić uwagę na następującą rzecz. Otóż, jak wspomniano wcześniej relacje są zbiorami, a zatem kolejność zwracania krotek przez SZBD i ich wyświetlania przez aplikację kliencką, jest dowolna. Przykładowe polecenie SELECT, oraz wynik jego działania, przedstawiono na slajdzie.

Zadanie (1)

- Odczytaj wszystkie dane z tabeli ETATY.

NAZWA	PLACA_OD	PLACA_DO
PROFESOR	3000	4000
ADIUNKT	2510	3000
ASYSTENT	1500	2100
DOKTORANT	800	1000
SEKRETARKA	1470	1650
DYREKTOR	4280	5100

Spróbuj napisać polecenie SQL, które spowoduje odczytanie wszystkich, niezmienionych, krotek z relacji ETATY. Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (1)

- Odczytaj wszystkie dane z tabeli ETATY.

```
SELECT * FROM ETATY;
```

Projekcja

SELECT imie, nazwisko **FROM** pracownicy;

IMIE	NAZWISKO
Jan	Marecki
Karol	Janicki
Pawel	Nowicki
Piotr	Nowak
Krzysztof	Kowalski
.....	.....

W większości przypadków nie ma konieczności odczytywania wszystkich atrybutów krotek z relacji. Najczęściej istotny jest tylko niewielki ich podzbiór. W takich sytuacjach można skorzystać z nieco bardziej skomplikowanej wersji polecenia SELECT, w której zamiast znaku „*” wymienia się listę nazw atrybutów, które mają się znaleźć w wyniku. Poszczególne atrybuty na tej liście są rozdzielane za pomocą przecinków. Proces wybierania atrybutów, które mają się znaleźć w wyniku nazywa się „projekcją”. Rozszerzona wersja polecenia SELECT wygląda następująco:

SELECT {atrybut1, atrybut2, ...} FROM {nazwa relacji};

Gdzie „atrybut” to nazwa atrybutu zdefiniowanego w relacji podanej za słowem kluczowym FROM. Przykładowo, polecenie:

SELECT imie, nazwisko FROM PRACOWNICY;

Odczyta z relacji PRACOWNICY wszystkie krotki, ale zwróci jedynie wartości dotyczące atrybutów IMIE i NAZWISKO. Wynik działania tego zapytania przedstawiono na slajdzie.

Zadanie (2)

- Odczytaj nazwę i adres wszystkich zespołów w tabeli ZESPOLY.

NAZWA	ADRES
ADMINISTRACJA	PIOTROWO 2
SYSTEMY ROZPROSZONE	PIOTROWO 3A
SYSTEMY EKSPERCKIE	STRZELECKA 14
ALGORYTMY	WIENIAWSKIEGO 16
BADANIA OPERACYJNE	MIELZYNSKIEGO 30

Spróbuj napisać polecenie SQL, które spowoduje odczytanie NAZWY i ADRESU z wszystkich krotek z relacji ZESPOLY. Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (2)

- Odczytaj nazwę i adres wszystkich zespołów w tabeli ZESPOLY.

```
SELECT nazwa, adres FROM zespoly;
```

Bazy danych

Wyrażenia

```
SELECT imie||' '||nazwisko, placa_pod*12 FROM pracownicy;
```

IMIE||''||NAZWISKO	PLACA_POD*12
Jan Marecki	56760
Karol Janicki	40200
Pawel Nowicki	36840
Piotr Nowak	47520
Krzysztof Kowalski	38760
.....	.....

Ćwiczenie 1 - podstawy (15)

W poleceniu SELECT nie trzeba koniecznie podawać samych nazw atrybutów, jak to było robione na poprzednich slajdach. Można również definiować pewne wyrażenia, które będą obliczane na kolejnych wartościach atrybutów (dla kolejnych przetwarzanych krotek). Na przykładzie przedstawionym na slajdzie, pokazano dwa z operatorów, które mogą zostać wykorzystane do konstrukcji wyrażeń, które można umieścić w klauzuli SELECT. Są to operatory '*' i '||'. Ten pierwszy operator to zwykły operator mnożenia. Prócz operatora mnożenia, na danych liczbowych (takich jak pensja), można również stosować operatory: dodawania ('+'), odejmowania ('-'), dzielenia ('/') i zmiany znaku ('-'), oraz operatory zmiany priorytetu ('(' i ')'). Stąd, wyrażenie PLACA_POD*12 oznacza, że dla każdej kolejno odczytywanej krotki reprezentującej pracownika z bazy danych, SZBD obliczy i zwróci jego roczną płacę (atrybut PLACA_POD reprezentuje płacę miesięczną). Z kolei operator '||' jest operatorem konkatenacji. W poleceniu przedstawionym na slajdzie wykorzystano go w wyrażeniu IMIE||' '||NAZWISKO. Użycie tego wyrażenia oznacza, że dla każdej kolejnej krotki SZBD wykonuje, i zwraca w relacji wynikowej, konkatenację imienia, spacji i nazwiska pracownika. Należy tutaj zwrócić uwagę na fakt, iż stałe typu łańcuchowego są w języku SQL otaczane apostrofami (np. stała ' ' w wyrażeniu IMIE||' '||NAZWISKO oznaczająca spację). W przeciwieństwie do stałych typu łańcuchowego, stałe typu liczbowego nie wymagają żadnych dodatkowych oznaczeń. Podsumowując, nieco bardziej rozbudowana wersja polecenia SELECT wygląda następująco:

SELECT {wyrażenie1, wyrażenie2,.....} FROM {nazwa relacji};

Gdzie wyrażeniem może być:

(w przypadku atrybutów typu liczbowego):

-nazwa atrybutu,

-stała liczbowa,

-suma, różnica, iloczyn, bądź iloraz dwóch wyrażeń,

w przypadku atrybutów typu łańcuchowego:

-nazwa atrybutu,

-stała łańcuchowa,

-konkatenacja dwóch wyrażeń.

Przykładowe polecenie przedstawione na slajdzie: „SELECT imie||' '||nazwisko, placa_pod*12 FROM pracownicy;” można zatem przetłumaczyć następująco: „Odczytaj wszystkie krotki z tabeli pracownicy; z każdej z tych krotek wyciągnij wartości atrybutów IMIE, NAZWISKO i PLACA_POD; IMIE i NAZWISKO skonkatenuj ze sobą i spacją (‘ ‘), a PLACE_POD pomnóż razy 12; wyniki tych operacji zwróć w relacji wynikowej.”.

Zadanie (3)

- Dla każdego pracownika oblicz jego dniówkę (1/20 płacy podstawowej) i wyświetl razem z jego imieniem i nazwiskiem.

IMIE	NAZWISKO	PLACA_POD/20
Jan	Marecki	236,5
Karol	Janicki	167,5
Pawel	Nowicki	153,5
Piotr	Nowak	198
Krzysztof	Kowalski	161,5
.....	.....	.....

Spróbuj napisać polecenie SQL, które spowoduje odczytanie z relacji PRACOWNICY imion i nazwisk pracowników, oraz obliczenie dniówki każdego z pracowników (1/20 płacy podstawowej). Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (3)

- Dla każdego pracownika oblicz jego dniówkę (1/20 płacy podstawowej) i wyświetl razem z jego imieniem i nazwiskiem.

```
SELECT imie, nazwisko, placa_pod/20
FROM pracownicy;
```

Bazy danych

Aliasy

```
SELECT imie||' '||nazwisko AS Pracownik,
 placa_pod*12 AS "Roczny dochód"
FROM pracownicy;
```

PRACOWNIK	Roczny dochód
Jan Marecki	56760
Karol Janicki	40200
Pawel Nowicki	36840
Piotr Nowak	47520
Krzysztof Kowalski	38760
.....	.....

Ćwiczenie 1 - podstawy (19)

Jak zapewne zauważyliście Państwo, nazwy atrybutów relacji wynikowej, kiedy w klauzuli SELECT podaje się bardziej skomplikowane wyrażenia niż tylko nazwa atrybutu, przypominają odpowiadające im wyrażenie (przykładowo, dla wyrażenia PLACA_POD*12, atrybut relacji wynikowej również nazywał się PLACA_POD*12). W przypadku bardziej skomplikowanych wyrażeń takie nazewnictwo kolumn przeszkadza przy analizie wyników zapytania w programie korzystającym z bazy danych. Można jednak ominąć ten problem nadając nowe nazwy (tzw. „aliasy") dowolnym wyrażeniom, które znajdą się w relacji wynikowej zapytania. Aliasy nie są obowiązkowe i najlepiej stosować je jedynie wyrażeń bardziej skomplikowanych niż sama nazwa atrybutu. Aby nadać wyrażeniu alias, należy za tym wyrażeniem użyć słowa kluczowego AS, a następnie podać alias. Podsumowując, rozszerzenie składni polecenia SQL o możliwość nadawania wyrażeniom aliasów wygląda następująco:

SELECT {wyrażenie1 [AS alias1], wyrażenie2 [AS alias2],.....} FROM {nazwa relacji};

Jeżeli przeanalizować przykład na slajdzie:

SELECT imie||' '||nazwisko AS Pracownik, placa_pod*12 AS "Roczny dochód" FROM pracownicy;

łatwo się zorientować, że jest to nieznacznie zmodyfikowany przykład z poprzednich slajdów, w którym wyrażeniu IMIE||' '||NAZWISKO nadawany jest alias Pracownik (AS Pracownik), a wyrażeniu PLACA_POD*12 alias "Roczny dochód". Dlaczego jednak pierwszy alias (Pracownik) nie jest otoczony cudzysłowami, a drugi alias ("Roczny dochód") jest? Otóż, kiedy alias nie jest otoczony cudzysłowami, nie można w nim umieszczać spacji, a co więcej wszystkie małe litery w aliasie są zamieniane na duże (zwróć uwagę na nazwę pierwszego atrybutu relacji wynikowej przedstawionej na slajdzie). Użycie cudzysłowów pozwala na używanie spacji oraz dużych i małych liter.

Zadanie (4)

- Dla każdego pracownika skonstruuj zdanie:

  „XXX pracuje na etacie YYY”

  gdzie XXX jest nazwiskiem pracownika a YYY nazwą jego etatu. Wykorzystaj operator konkatenacji poznany poprzednio. Skonstruowanemu przez siebie wyrażeniu nadaj alias ZDANIE.

ZDANIE
Marecki pracuje na etacie DYREKTOR
Janicki pracuje na etacie PROFESOR
.....

Spróbuj napisać polecenie SQL, które spowoduje odczytanie z relacji PRACOWNICY nazwisk i etatów pracowników, oraz przeprowadzi konstrukcję zdań typu „XXX pracuje na etacie YYY’, gdzie XXX to nazwisko pracownika a YYY jego etat. Do konstrukcji zdań należy wykorzystać operator konkatenacji. Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (4)

- Dla każdego pracownika skonstruuj zdanie:

  „XXX pracuje na etacie YYY”

  gdzie XXX jest nazwiskiem pracownika a YYY nazwą jego etatu. Wykorzystaj operator konkatenacji poznany poprzednio. Skonstruowanemu przez siebie wyrażeniu nadaj alias ZDANIE.

```
SELECT nazwisko||' pracuje na etacie '||etat AS zdanie
FROM pracownicy;
```

Wartości puste

```
SELECT nazwisko, placa_dod FROM pracownicy;
```

NAZWISKO	PLACA_DOD
Marecki	980,5
Nowicki	
.....	.....

```
SELECT nazwisko, NVL(placa_dod,0) FROM pracownicy;
```

NAZWISKO	PLACA_DOD
Marecki	980,5
Nowicki	0
.....	.....

Przeanalizujmy pierwsze zapytanie przedstawione na slajdzie:

SELECT nazwisko, placa_dod FROM pracownicy;

Zgodnie z tym co powiedziano dotychczas, zapytanie powinno odczytać wszystkie krotki z tabeli PRACOWNICY, z odczytanych krotek pobrać wartości atrybutów: NAZWISKO i PLACA_DOD, i zwrócić je w postaci relacji wynikowej. Jeżeli jednak spojrzeć na wynik zapytania przedstawiony na slajdzie, można zauważyć jedną rzecz. Otóż, dla pracownika o nazwisku Nowicki, w tabeli przedstawiającej relację wynikową, nie zwrócono żadnej wartości w kolumnie PLACA_DOD. Aby wytłumaczyć przyczynę tego zjawiska należy wprowadzić koncepcję wartości „pustej” oznaczanej jako NULL. Wartość NULL jest specjalną, wyróżnioną wartością, którą można przypisać dowolnemu atrybutowi dowolnego typu w dowolnej krotce. Oznacza ona, że faktyczna wartość tego atrybutu jest niedostępna, nieprzypisana, nieznana lub nieistotna. Wartość NULL najczęściej nie jest wyświetlana w ogóle, bądź oznaczana w wynikach jako NULL. W przypadku przykładowego zapytania, pracownik Nowicki nie ma po prostu płacy dodatkowej, co zostało odwzorowane w jego krotce tym, że zapisano wartość NULL w atrybucie PLACA_DOD.

Przy postępowaniu z wartościami NULL należy zachować daleko posuniętą ostrożność. Jeżeli w jakimkolwiek wyrażeniu wystąpi wartość NULL, to wartość tego wyrażenia również wynosi NULL. Jest to o tyle intuicyjne, że wynik obliczeń na wartości nieznanej również jest nieznany. Aby poradzić sobie z tym problemem można założyć jakąś konkretną wartość odpowiadającą wartości NULL. Przykładowo, w przypadku płacy dodatkowej pracownika, można założyć, że wartość NULL (nie posiada płacy dodatkowej) odpowiada wartości 0 (płaca dodatkowa pracownika wynosi 0). Aby wykonać taką zamianę należy wykorzystać specjalną funkcję służącą do tego celu - funkcję NVL.

Funkcja NVL przyjmuje 2 parametry i działa w następujący sposób: jeżeli pierwszy parametr ma wartość różną od NULL to zwraca tą wartość, w przeciwnym wypadku zwraca wartość drugiego parametru. Aby zobaczyć przykład zastosowania tej funkcji można przeanalizować drugi przykład pokazany na slajdzie.

```
SELECT nazwisko, NVL(placa_dod,0) FROM pracownicy;
```

Jak łatwo zauważyć, nazwę atrybutu PLACA_DOD zastąpiono wyrażeniem NVL(PLACA_DOD,0). Zgodnie z wcześniejszym opisem działania funkcji NVL, jeżeli PLACA_DOD będzie zawierać jakąś liczbę, to funkcja NVL zwróci tą liczbę. Jeżeli PLACA_DOD będzie zawierać NULL, to zostanie zwrócona wartość drugiego parametru, czyli 0. Analizując wynik tego zapytania można łatwo zauważyć, że tym razem dla pracownika Nowickiego podano wartość płacy dodatkowej równą zero. Funkcję NVL można stosować samodzielnie (tak jak na przykładzie), ale również jako fragment wyrażenia np. a+NVL(x,200).

Zadanie (5)

- Dla każdego pracownika oblicz jego roczną płacę z uwzględnieniem płacy dodatkowej. W wyniku ma się znaleźć nazwisko i obliczona roczna płaca pracownika. Wyrażeniu obliczającemu roczną płacę nadaj alias „DOCHOD”.

NAZWISKO	DOCHOD
Marecki	68526
Janicki	47520
Nowicki	36840
.....	.....

Spróbuj napisać polecenie SQL, które spowoduje odczytanie z relacji PRACOWNICY wszystkie krotki i dla każdego pracownika obliczy jego całkowity roczny dochód (z uwzględnieniem płacy dodatkowej). Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (5)

- Dla każdego pracownika oblicz jego roczną płacę z uwzględnieniem płacy dodatkowej. W wyniku ma się znaleźć nazwisko i obliczona roczna płaca pracownika. Wyrażeniu obliczającemu roczną płacę nadaj alias „DOCHOD”.

```
SELECT
 nazwisko,
 12*(placa_pod+nv;(placa_dod,0)) AS dochod
FROM pracownicy;
```

Bazy danych

Eliminacja duplikatów

SELECT DISTINCT etat **FROM** pracownicy;

ETAT
ADIUNKT
ASYSTENT
DOKTORANT
DYREKTOR
PROFESOR
SEKRETARKA

Ćwiczenie 1 - podstawy (26)

Przy realizacji zapytań może się zdarzyć, że w relacji wynikowej znajdzie się kilka krotek o takich samych wartościach na wszystkich odczytanych atrybutach. Przykładowo, jeżeli wykonamy zapytanie:

SELECT etat FROM pracownicy;

Niektóre etaty mogą w relacji wynikowej pojawić się wielokrotnie. W takiej sytuacji najczęściej możemy chcieć usunąć duplikaty pozostawiając jedynie po jednym egzemplarzu każdej krotki. Do przeprowadzenia takiej operacji służy klauzula DISTINCT, którą należy podać po słowie kluczowym SELECT. Jeżeli klauzula DISTINCT pojawi się po SELECT, SZBD usunie wszystkie duplikaty krotek z relacji wynikowej.
Podsumowując, rozszerzona składnia polecenia SELECT wygląda następująco:

SELECT [DISTINCT] {wyrażenie1 [AS alias1], wyrażenie2 [AS alias2],.....} FROM {nazwa relacji};

Przykładowe polecenie przedstawione na slajdzie:

SELECT DISTINCT etat FROM pracownicy;

znajduje wszystkie różne nazwy etatów na których zatrudnieni są pracownicy. Należy tutaj zwrócić uwagę na jedną rzecz. Teoretycznie ten sam wynik moglibyśmy uzyskać wykonując proste zapytanie do relacji etaty:

SELECT nazwa FROM etaty;

Czym zatem różnią się oba powyższe polecenia? Otóż, pierwsze polecenie zwróci wszystkie etaty na których zatrudnieni są jacyś pracownicy, podczas gdy drugie zwróci wszystkie etaty zdefiniowane w bazie danych, nawet takie, na których nie jest zatrudniony żaden pracownik.

Zadanie (6)

- Znajdź listę identyfikatorów zespołów, w których są zatrudnieni pracownicy. W wyniku identyfikatory zespołów nie mogą się powtórzyć.

ID_ZESP
10
20
30
40
(null)

Spróbuj napisać polecenie SQL, które spowoduje odczytanie z relacji PRACOWNICY wszystkich krotek, wybranie z tych krotek wartości oznaczających numer zespołu, do którego należą poszczególni pracownicy, oraz usunięcie powtarzających się wartości wśród tych identyfikatorów i zwrócenie pozostałych wartości w relacji wynikowej. Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (6)

- Znajdź listę identyfikatorów zespołów, w których są zatrudnieni pracownicy. W wyniku identyfikatory zespołów nie mogą się powtórzyć.

```
SELECT DISTINCT id_zesp FROM pracownicy;
```

Bazy danych

Sortowanie wyników zapytania

```
SELECT nazwisko FROM pracownicy
ORDER BY nazwisko;
```

NAZWISKO
Dolny
Grzybowska
Janicki
Kotarski
Kotlarczyk
Kowalski
.....

Ćwiczenie 1 - podstawy (29)

Przy okazji omawiania podstawowej składni polecenia SELECT wspomniano, że wyniki zapytania są zwracane w przypadkowej kolejności. Czasem jednak możemy być zainteresowani otrzymaniem ich w jakiejś określonej przez nas kolejności. W tym celu należy zażądać od SZBD, aby, przed zwróceniem wyników zapytania, posortował je według wartości dowolnego, zdefiniowanego przez nas wyrażenia. Robi się to za pomocą klauzuli ORDER BY dodawanej na końcu polecenia. Za klauzulą ORDER BY podaje się listę wyrażeń, lub aliasów wyrażeń (zdefiniowanych przy klauzuli SELECT) oddzielonych przecinkami. Wynik zapytania zostanie posortowany według wartości tych wyrażeń.

Rozszerzona o klauzulę ORDER BY składnia polecenia SELECT wygląda następująco:

SELECT [DISTINCT] {wyrażenie1 [AS alias1], wyrażenie2 [AS alias2],.....} FROM {nazwa relacji}

ORDER BY {wyrażenie3 [ASC|DESC], wyrażenie4 [ASC|DESC], alias1 [ASC|DESC], alias2 [ASC|DESC].....};

W celu wyjaśnienia działania klauzuli ORDER BY zostanie omówionych kilka przykładowych zapytań:

1. SELECT nazwisko FROM PRACOWNICY ORDER BY nazwisko;

Początek zapytania można już wytłumaczyć na podstawie tego co dotychczas omówiono. SELECT nazwisko FROM pracownicy... znaczy: odczytaj z relacji PRACOWNICY wszystkie krotki, odczytaj z nich atrybut NAZWISKO i zwróć odczytane wartości w relacji wynikowej. Końcówka ORDER BY mówi SZBD, aby posortować otrzymane krotki rosnąco według nazwisk (porządek leksykograficzny).

Analizując podaną powyżej ogólną składnię polecenia SELECT można zauważyć, że za wyrażeniem w klauzuli ORDER BY można opcjonalnie podać słowo kluczowe ASC bądź DESC. Słowo to określa porządek sortowania. ASC oznacza sortowanie rosnąco i jest domyślne. DESC oznacza sortowanie malejąco. Przykładowo, polecenie

SELECT nazwisko FROM pracownicy ORDER BY nazwisko ASC;

zwróci te same wyniki i w tej samej kolejności, co zapytanie 1. Z kolei zapytanie

SELECT nazwisko FROM pracownicy ORDER BY nazwisko DESC;

zwróci wyniki w odwrotnym porządku w stosunku do poprzednich zapytań.

2. SELECT nazwisko FROM pracownicy ORDER BY placa_pod;

Zapytanie 2 podobnie jak poprzednie zapytania zwraca wszystkie nazwiska pracowników zapisane w relacji PRACOWNICY. Tym razem jednak wyniki są sortowane według atrybutu PLACA_POD. Jak łatwo zatem można zauważyć, sortować można również według atrybutów, które nie są wymienione w klauzuli SELECT. Jest to możliwe, gdyż projekcja odbywa się dopiero po posortowaniu krotek otrzymanych w wyniku zapytania.

3. SELECT nazwisko FROM pracownicy ORDER BY placa_pod/20;

Niniejsze zapytanie odczytuje wszystkie krotki z relacji PRACOWNICY, oblicza dzienną płacę każdego pracownika, sortuje otrzymane krotki według obliczonych wartości dziennej płacy i ostatecznie odczytuje z posortowanych krotek atrybut nazwisko i zwraca go w relacji wynikowej. W klauzuli ORDER BY można zatem podawać nie tylko nazwy kolumn, ale również wyrażenia.

4. SELECT nazwisko,placa_pod/20 AS dniowka FROM pracownicy ORDER BY dniowka;

W niniejszym zapytaniu wyrażenie obliczające dzienną płacę pracownika umieszczono w klauzuli SELECT żądając tym samym, aby, prócz nazwiska, w relacji wynikowej znalazła się jego wartość. Warto również zauważyć, że wyrażeniu temu nadano alias DNIOWKA, oraz że po klauzuli ORDER BY użyto właśnie tego aliasu. W zapytaniu tym wyniki są również sortowane według wartości dziennej płacy pracowników. Można zatem, zamiast wyrażenia, umieścić w klauzuli ORDER BY jego alias.

5. SELECT nazwisko, etat, placa_pod FROM pracownicy ORDER BY etat, placa_pod;

Niniejsze zapytanie odczytuje z pobranych z relacji PRACOWNICY krotek nazwisko, etat i płacę podstawową pracownika. W klauzuli ORDER BY umieszczono dwa proste wyrażenia, oddzielone od siebie przecinkami. Sortowanie dla tak sformułowanego polecenia odbywa się w następujący sposób. Najpierw krotki są sortowane według atrybutu ETAT. W ramach zbioru krotek, o takiej samej wartości atrybutu ETAT, krotki są sortowane według atrybutu PLACA_POD.

W klauzuli ORDER BY można podać dowolną liczbę wyrażeń, przy czym każde kolejne jest wykorzystywane do posortowania krotek, dla których wszystkie poprzednie wyrażenia mają taką samą wartość.

Porządek sortowania zależy od typu sortowanych danych i wygląda następująco (porządek domyślny, nie zmodyfikowany za pomocą słowa kluczowego DESC).

liczby – od mniejszych do większych

daty – od wcześniejszych do późniejszych

łańcuchy znaków – alfabetycznie

wartości puste – w zależności od SZBD (najczęściej są wymieniana jako pierwsze albo ostatnie).

Zadanie (7)

- Zastanów się co robi następujące zapytanie:

```
SELECT
 nazwisko||' '||etat,
 placa_pod*12 AS dochod
FROM pracownicy
ORDER BY etat DESC, imie, dochod DESC;
```


Spróbuj się zastanowić jakie jest działanie zapytania przedstawionego na slajdzie. Kiedy będziesz pewien/pewna odpowiedzi porównaj z opisem przedstawionym na kolejnym slajdzie.

Rozwiązanie (7)

```
SELECT
 nazwisko||' '||etat,
 placa_pod*12 AS dochod
FROM pracownicy
ORDER BY etat DESC, imie, dochod DESC;
```

Zapytanie odczytuje wszystkie krotki z relacji PRACOWNICY. Oblicza dwa wyrażenia: konkatenację nazwiska, spacii i etatu pracownika oraz roczną płacę pracownika. Drugiemu z tych wyrażeń nadaje alias DOCHOD. Wartości tych wyrażeń trafiają do relacji wynikowej. Po utworzeniu relacji wynikowej, krotki są w niej sortowane według etatu malejąco. Wszystkie krotki o takiej samej wartości etatu są następnie sortowane rosnąco według imienia. Wszystkie krotki o takiej samej wartości etatu i imienia są sortowane malejąco według dochodu, gdzie dochód jest aliasem wyrażenia z klauzuli select i reprezentuje roczną płacę pracownika.

Bazy danych

Selekcja

```
SELECT nazwisko  FROM pracownicy
WHERE etat='PROFESOR'
ORDER BY nazwisko;
```

NAZWISKO
Janicki
Kowalski
Nowak
Nowicki

Ćwiczenie 1 - podstawy (34)

We wszystkich dotychczasowych przykładach zapytania odczytywały wszystkie krotki z zadanej relacji w bazie danych. Najczęściej jednak aplikacje, które wykorzystują bazy danych do składowania swoich danych, potrzebują jednorazowo odczytać jedynie niewielki podzbiór krotek zapisanych w relacjach. W celu wybrania, które krotki mają się znaleźć w relacji wynikowej, stosuje się klauzulę WHERE. Za klauzulą WHERE podaje się warunek, zdefiniowany na wartościach atrybutów w relacji, który musi być spełniony, aby krotka znalazła się w relacji wynikowej. Operację wyboru krotek, które mają się znaleźć w rozwiązaniu nazywa się „selekcją". Składnia polecenia SELECT, rozszerzona o klauzulę WHERE, wygląda następująco:

SELECT [DISTINCT] {wyrażenie1 [AS alias1], wyrażenie2 [AS alias2],.....} FROM {nazwa relacji}

WHERE warunek_elementarny

ORDER BY {wyrażenie5 [ASC|DESC], wyrażenie6 [ASC|DESC], alias1 [ASC|DESC], alias2 [ASC|DESC].....};

gdzie „warunek_elementarny", to porównanie jednego, dwóch lub większej liczby wyrażeń za pomocą odpowiednich operatorów logicznych.

Przykładowe zapytanie przedstawione na slajdzie:

SELECT nazwisko, etat FROM pracownicy WHERE etat='PROFESOR' ORDER BY nazwisko;

można przetłumaczyć zatem następująco: odczytaj z relacji PRACOWNICY wszystkie krotki, w których atrybut ETAT ma wartość równą 'PROFESOR'. Posortuj odczytane krotki według wartości atrybutu NAZWISKO i zwróć wartości atrybutu NAZWISKO z odczytanych i posortowanych krotek. W praktyce polecenie to zwróci posortowaną alfabetycznie listę nazwisk wszystkich profesorów.

W zapytaniu tym, warunek elementarny ma postać ETAT=‘PROFESOR’ gdzie ETAT i ‘PROFESOR’ są wyrażeniami a ‘=‘ jest operatorem logicznym.

Język SQL dysponuje dużym zbiorem operatorów logicznych, które można wykorzystać przy konstrukcji warunków. Operatory te zostały przedstawione na kolejnych slajdach.

Selekcja – operatory logiczne

=, !=, <>, >, >=, <, <=

```
SELECT nazwisko, placa_pod, etat
FROM pracownicy
WHERE placa_pod > 400;
```

```
SELECT nazwisko, id_zesp
FROM pracownicy
WHERE
 placa_dod > (placa_pod/10);
```

```
SELECT id_prac, nazwisko, etat
FROM pracownicy
WHERE etat!= 'ASYSTENT';
```


Przy tworzeniu warunków w klauzuli WHERE można stosować znane z różnych języków programowania operatory binarne porównujące dwie wartości. Są to operatory testujące czy dwie liczby są równe (‘=‘), różne (‘!=‘,’<>’), czy jedna z liczb jest większa albo większa lub równa drugiej (‘>’,’>=‘) oraz czy jedna z liczb jest mniejsza, albo mniejsza lub równa drugiej (‘<‘,’<=‘). Operatory te można stosować na liczbach, łańcuchach (porządek alfabetyczny) oraz datach (data wcześniejsza jest mniejsza).

Selekcja – operatory logiczne – cd.

BETWEEN ... AND ...

```
SELECT nazwisko, placa_pod, etat
FROM pracownicy
WHERE placa_pod BETWEEN 208 AND 1070;
```

IN

```
SELECT nazwisko, placa_pod, id_zesp
FROM pracownicy
WHERE etat IN ('PROFESOR', 'DYREKTOR');
```


Operator BETWEEN AND sprawdza, czy jedna wartość zawiera się pomiędzy dwoma innymi (włącznie). Składnia operatora wygląda następująco:

x BETWEEN y AND z

gdzie ‘x’ to sprawdzana wartość, ‘y’ to wartość określająca początek przedziału a ‘z’ koniec. Pod ‘x’, ‘y’ i ‘z’ można podstawić dowolne wyrażenia, przy czym wymagane jest aby ‘y’ było mniejsze od ‘z’. Wartości wyrażeń podstawianych pod ‘x’, ‘y’ albo ’z’ mogą być zarówno liczbami jak i łańcuchami oraz datami. Ważne jest jednak aby wszystkie były tego samego typu. Przeanalizujmy klauzulę WHERE pierwszego przykładu:

WHERE placa_pod BETWEEN 208 AND 1070

Oznacza ona, że poszukiwane są wszystkie krotki, w których wartość atrybutu PLACA_POD mieści się pomiędzy 208 a 1070.

Operator IN sprawdza czy jedna wartość jest równa przynajmniej jednej z wartości wymienionych na liście. Składnia operatora wygląda następująco:

x IN (a_1, a_2, a_3,...,a_n),

gdzie „x” to sprawdzana wartość, a „a_i” to wartości z którymi porównywana jest wartość „x”. Pod „x” oraz „a_i” można podstawić dowolne wyrażenia. „x” oraz „a_i” mogą być liczbami, łańcuchami oraz datami, ważne jest jednak aby były tego samego typu. Przeanalizujmy klauzulę WHERE drugiego przykładu:

WHERE etat IN ('PROFESOR', 'DYREKTOR')

Oznacza ona, że poszukiwane są wszystkie krotki, w których wartość atrybutu ETAT jest równa ‘PROFESOR’ albo ‘DYREKTOR’.

Selekcja – operatory logiczne – cd.

LIKE

```
SELECT nazwisko, placa_pod, id_zesp
FROM pracownicy
WHERE nazwisko LIKE 'M%';
```

IS NULL

~~**SELECT** nazwisko, placa_pod~~
~~**FROM** pracownicy~~
~~**WHERE** placa_dod = **NULL**;~~

```
SELECT nazwisko, placa_pod
FROM pracownicy
WHERE placa_dod IS NULL;
```


Operator LIKE jest specjalnym operatorem stosowanym do złożonego porównywania łańcuchów. Składnia operatora LIKE to:

x LIKE maska,

gdzie x jest dowolnym wyrażeniem typu łańcuchowego, a maska jest specjalnym łańcuchem zawierającym normalne znaki oraz znaki o specjalnym znaczeniu. Znaki specjalne to: ‘%’ i ‘_’, gdzie ‘%’ oznacza dowolny ciąg znaków (zero lub więcej) a ‘_’ dokładnie jeden, dowolny, znak. Operator LIKE sprawdza, czy zadany łańcuch spełnia warunki zdefiniowane przez maskę. Rozważmy przykładowe zapytanie przedstawione na slajdzie. Klauzula WHERE w tym zapytaniu zawiera warunek: „nazwisko LIKE 'M%‘”. W warunku tym maska ma postać ‘M%’. Reprezentuje ona wszystkie łańcuchy, które jako pierwszy znak mają literę M, po której może się znajdować dowolny ciąg znaków. Warunek ten będzie zatem spełniony, jeśli łańcuch zapisany w atrybucie NAZWISKO będzie się zaczynał od litery ‘M’. Poniżej przedstawiono kilka innych przykładów maski:

‘%SKI’ – wszystkie łańcuchy kończące się na SKI (np.. ‘KOTARSKI’).

‘MALINOWSK_’ – wszystkie łańcuchy zaczynające się od MALINOWSK i mające dowolny jeden znak na końcu (np.. ‘MALINOWSKI’, ‘MALINOWSKA’).

Operator IS NULL sprawdza, czy wartość danego wyrażenia jest równa NULL. W przykładowym zapytaniu sprawdza on, czy wartość atrybutu PLACA_DOD wynosi NULL. Należy w tym miejscu zwrócić szczególną uwagę na to, iż sprawdzanie, czy dany atrybut ma wartość NULL za pomocą operatora ‘=‘ nie ma sensu. Wynika to z faktu, że NULL stanowi wartość nieznaną. Wynik porównania wartości nieznanej z jakąś inną wartością jest również nieznany. Różnica w semantyce pomiędzy dwoma przykładowymi warunkami: placa_dod=NULL i placa_dod IS NULL jest zatem znaczna.

Pierwsze wyrażenie można przetłumaczyć następująco: czy placa_dod jest równa wartości nieznanej? Drugie wyrażenie można przetłumaczyć: czy placa_dod jest wartością nieznaną? Na pierwsze pytanie nie można odpowiedzieć, na drugie można. Jeżeli zostanie użyty operator ‘=‘ zamiast IS NULL, SZBD nie będzie mógł stwierdzić czy warunek jest prawdziwy i w rezultacie żadne krotki nie zostaną zwrócone.

Zanegowane operatory logiczne

NOT BETWEEN ... AND ...
NOT IN
NOT LIKE
IS NOT NULL

```
SELECT nazwisko, placa_pod, id_zesp
FROM pracownicy
WHERE etat NOT IN ('PROFESOR', 'DYREKTOR');
```

```
SELECT nazwisko, etat, placa_pod+NVL(placa_dod)
FROM pracownicy
WHERE nazwisko NOT LIKE '%SKI';
```


W standardzie SQL istnieją również operatory stanowiące negację operatorów przedstawionych poprzednio. Są to operatory:

- NOT BETWEEN AND – wartość NIE znajduje się w zadanym przedziale
- NOT IN – wartość NIE znajduje się na zadanej liście
- NOT LIKE – łańcuch NIE pasuje do maski
- IS NOT NULL – wartość wyrażenia NIE jest nieznana.

Zadanie (8)

- Podaj nazwiska i miesięczną płacę pracowników (z uwzględnieniem płacy dodatkowej), którzy mają jakąś płacę dodatkową.

NAZWISKO	PLACA_POD+PLACA_DOD
Marecki	5710,5
Janicki	3960
Kowalski	4035
Opolski	2320,2
.....	.....

Spróbuj napisać polecenie SQL, które spowoduje odczytanie z relacji PRACOWNICY krotek wszystkich pracowników, którzy dostają płacę dodatkową. Wyświetl nazwiska tych pracowników i ich miesięczną płacę z uwzględnieniem płacy dodatkowej. Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (8)

- Podaj nazwiska i miesięczną płacę pracowników (z uwzględnieniem płacy dodatkowej), którzy mają jakąś płacę dodatkową.

```
SELECT
 nazwisko,
 placa_pod+placa_dod
FROM pracownicy
WHERE placa_dod IS NOT NULL;
```


Zwróć uwagę, że nie zastosowano tutaj funkcji NVL. Nie było to konieczne, gdyż z warunku selekcji wynika, że PLACA_DOD w wyniku nigdy nie będzie miała wartości NULL.

Selekcja – złożone warunki

```
SELECT nazwisko
FROM pracownicy
WHERE etat='PROFESOR' AND placa_pod > 1000;
```

NAZWISKO
Janicki
Nowicki
Nowak
Kowalski

Warunki w klauzuli WHERE można w dowolny sposób komplikować używając operatorów AND, OR i NOT. Operator NOT ma najwyższy priorytet. Operator AND ma wyższy priorytet niż operator OR. Aby zmienić priorytet operatorów można użyć nawiasów. Podsumowując, rozbudowana składnia polecenia SELECT wygląda następująco:

SELECT [DISTINCT] {wyrażenie1 [AS alias1], wyrażenie2 [AS alias2],.....} FROM {nazwa relacji}

WHERE warunek_złożony

ORDER BY {wyrażenie3 [ASC|DESC], wyrażenie4 [ASC|DESC], alias1 [ASC|DESC], alias2 [ASC|DESC].....};

Gdzie „warunek_złożony” to:

- warunek_elementarny,
- (warunek_złożony),
- NOT warunek_złożony,
- warunek_złożony OR warunek_złożony,
- warunek_złożony AND warunek_złożony,

a warunek elementarny to:

wyrażenie1 operator_logiczny wyrażenie2.

Przykładowe zapytanie przedstawione na slajdzie:

```
SELECT nazwisko
FROM pracownicy
WHERE etat='PROFESOR' AND placa_pod > 1000;
```

można przetłumaczyć zatem następująco: odczytaj z relacji PRACOWNICY wszystkie krotki, dla których wartość atrybutu etat jest równa stałej ‘PROFESOR’, a wartość atrybutu PLACA_POD jest większa od stałej 1000, z odczytanych krotek odczytaj wartości atrybutu NAZWISKO i zwróć je w relacji wynikowej. W praktyce znaczy to: znajdź nazwiska wszystkich profesorów zarabiających powyżej 1000 złotych.

Selekcja – złożone warunki – cd.

	TRUE	FLASE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Przy okazji omawiania operatora IS NULL wspomniano o problemach przy porównywaniu wartości atrybutów z atrybutami o nieznanej wartości (NULL), bądź stałą NULL. Porównanie dwóch wyrażeń, z których przynajmniej jedno ma wartość NULL, daje w wyniku porównania wartość logiczną UNKNOWN – nieznany wynik. Tabele wartości logicznych z uwzględnieniem wartości UNKNOWN, przedstawiono na slajdzie. Wartość logiczna UNKNOWN, podobnie jak wartość logiczna FALSE, usuwa krotkę z potencjalnego zbioru rozwiązań.

Zadanie (9)

- Znajdź wszystkich pracowników, którzy pracują na etacie ASYSTENT albo SEKRETARKA i zarabiają mniej niż 900 złotych lub takich pracowników, którzy posiadają pensję dodatkową. Dla każdego ze znalezionych pracowników wyświetl jego nazwisko.

NAZWISKO
Marecki
Janicki
Kowalski
.....

Spróbuj napisać polecenie SQL, które spowoduje odczytanie z relacji PRACOWNICY krotek wszystkich pracowników, którzy albo dostają płacę dodatkową albo zarabiają mniej niż 900 zł i są albo ASYSTENTEM albo SEKRETARKĄ. Wyświetl nazwiska tych pracowników. Kiedy to zrobisz, sprawdź, czy się nie pomyliłeś/łaś porównując Twoje rozwiązanie z rozwiązaniem przedstawionym na kolejnym slajdzie.

Rozwiązanie (9)

- Znajdź wszystkich pracowników, którzy pracują na etacie ASYSTENT albo SEKRETARKA i zarabiają mniej niż 900 złotych lub takich pracowników, którzy posiadają pensję dodatkową. Dla każdego ze znalezionych pracowników wyświetl jego nazwisko.

```
SELECT nazwisko
FROM pracownicy
WHERE etat IN ('ASYSTENT', 'SEKRETARKA')
 AND placa_pod<900
 OR placa_dod IS NOT NULL;
```

Podsumowanie

```
SELECT [DISTINCT]
 {wyrażenie1 [AS alias1],
  wyrażenie2 [AS alias2],.....}
FROM {nazwa_relacji}
WHERE {warunek_złożony}
ORDER BY
 {wyrażenie3 [ASC|DESC],
  wyrażenie4 [ASC|DESC],
  alias1 [ASC|DESC],
  alias2 [ASC|DESC].....};
```


Na tym ćwiczeniu zapoznaliście się Państwo z podstawową składnią polecenia SELECT. Poznaliście następujące klauzule polecenia SELECT:

- SELECT służąca do określenia jakie atrybuty mają się znaleźć w relacji wynikowej (projekcja),
- DISTINCT pozwalającą na usunięcie duplikatów z relacji wynikowej,
- FROM pozwalającą na określenie z której relacji należy pobrać krotki,
- WHERE pozwalającą zdefiniować warunki jakie muszą spełniać zwracane krotki (selekcja),
- ORDER BY pozwalającą na posortowanie relacji wynikowej przed przesłaniem jej do klienta.

Poznaliście również operatory, których można użyć przy definiowaniu warunków selekcji i dowiedzieliście się jak można sobie radzić w wartością nieznaną (NULL).